Schedule Generation Schemes and Genetic Algorithm for the Scheduling Problem with Skilled Operators and Arbitrary Precedence Relations

Raúl Mencía[1] and **María R. Sierra**[1] and **Carlos Mencía**[2] and **Ramiro Varela**[1]
[1]Department of Computer Science,
University of Oviedo, 33204 Gijón (Spain)
e-mail: raul89@gmail.com, {sierramaria, ramiro}@uniovi.es
[2]CASL, University College Dublin, Ireland
e-mail: carlos.mencia@ucd.ie

Abstract

In real-life production environments it is often the case that the processing of a task on a given machine requires the assistance of a human operator specially skilled to process that task. In this paper, we tackle a scheduling problem involving operators that are skilled to manage only subsets of the whole set of tasks in a given shop floor. This problem was recently proposed motivated by a handicraft company. In order to solve it, we make some contributions. We first propose a general schedule builder and particularize it to generate several complete solution spaces. This schedule builder is then exploited by a genetic algorithm that incorporates a number of problem-specific components, including a coding schema as well as crossover and mutation genetic operators. An experimental study shows substantial improvements over existing methods in the literature and reveals useful insights of practical interest.

Introduction

The scheduling problem with arbitrary precedence relations and skilled operators, defined in (Agnetis, Murgia, and Sbrilli 2014) and denoted JSSO, is a generalization of the job shop scheduling problem with operators (JSO) defined in (Agnetis et al. 2011), which in turn generalizes the classic job shop scheduling problem (JSP). In the JSP each task belongs to one and only one job, and each job defines a linear ordering for the processing of its tasks. The JSO extends the JSP in such a way that each task must be assisted by a human operator, the number of them being lower than both the number of jobs and the number of machines. In the JSO all the operators are equally skilled to assist any task, what may be unrealistic in many production environments.

The JSSO is motivated by a real-life handicraft company where the operators have different expertise. For example, an apprentice may be able to perform only a limited subset of single assembly tasks while the most difficult operations may require more experienced workers. Therefore, finding good solutions for the JSSO is an issue of major interest for human resources management as it would allow the company to make the best use of its employees and to plan their training for future projects.

To solve the JSSO, two heuristic algorithms were proposed in (Agnetis, Murgia, and Sbrilli 2014), which were evaluated across a benchmark set defined in accordance with the characteristics of the problems of the handicraft company. These algorithms are termed STA-OMSB and MSB-DOS respectively, and both of them exploit the well-known shifting bottleneck heuristic (Adams, Balas, and Zawack 1988). The results of these algorithms were compared with those obtained by Cplex MILP solver from a formulation proposed in the same paper.

In this paper we make several contributions towards understanding and solving the JSSO problem. Firstly, we formalize it using a disjunctive graph model, which makes it easier to reason about the problem. Then, we explore different ways of obtaining reduced dominant solution spaces. Concretely, we propose a general schedule builder, termed $SOG\&T$, and particularize it in three different ways. The hardness of the JSSO motivates the use of approximate algorithms. Accordingly, we propose a genetic algorithm that exploits the $SOG\&T$ and incorporates a novel problem-dependent coding scheme. The results from an experimental study indicate that our approach is very effective and outperforms previous methods in the state of the art for this problem.

The remaining of the paper is organized as follows. In the next two sections, we give a formal definition of the JSSO as a constraint satisfaction problem with optimization and propose a disjunctive graph model to represent problem instances and schedules. Then, we describe and analyze the proposed schedule generation scheme termed $SOG\&T$. After that, we introduce the genetic algorithm devised to solve the JSSO and summarize the results of an experimental study where the genetic algorithm is evaluated and compared with the state of the art. The paper finishes with some conclusions and ideas for future work.

Problem Formulation

In the scheduling problem with skilled operators and arbitrary precedence relations, we are given a set $\mathcal{M}$ of q machines, a set $\mathcal{O}$ of p operators and a set $\mathcal{T}$ of n tasks or operations. The task u requires two resources during its processing time p_u: a particular machine $m_u \in \mathcal{M}$ and one of the operators $o \in \mathcal{O}_u \subseteq \mathcal{O}$, where $\mathcal{O}_u$ denotes the subset of operators skilled to assist the task u. We assume that the

processing time p_u is independent of the assisting operator.

There are arbitrary precedence relations among tasks specified by a $task\ graph$ $(\mathcal{T}, E)$ where nodes correspond to tasks and an arc $(u, v) \in E$ means that task u must be performed before task v starts.

The objective is to allocate a starting time st_u and an operator o_u to each task $u \in \mathcal{T}$, such that the makespan is minimized and the following constraints are satisfied:

i. The tasks must be processed following the order expressed by the task graph; i.e., $st_u + p_u \leq st_v$ if $(u, v) \in E$.

ii. Each task u must be assisted by a skilled operator to do it; i.e., $o_u \in \mathcal{O}_u$, and the operator is allocated to the task over its whole processing time.

iii. Two tasks assisted by the same operator or processed on the same machine cannot overlap; i.e., if $(o_u = o_v) \vee (m_u = m_v)$ then $(st_u + p_u \leq st_v) \vee (st_v + p_v \leq st_u)$.

iv. The tasks cannot be preempted; i.e., $C_u = st_u + p_u$, where C_u denotes the completion time of task u.

This problem was firstly defined in (Agnetis, Murgia, and Sbrilli 2014) and denoted JSSO. In some cases, the task graph has a tree structure with the root corresponding to the final assembly task. However, the topology of the task graph may be different and, for example, it may represent a number of independent sequences of operations which are then termed jobs. The JSSO problem is a generalization of the JSP and JSO, and can be seen as a particular case of the Multi-mode Resource Constrained Project Scheduling Problem (MRCPSP).

A Disjunctive Graph Model

Scheduling problems are usually represented by means of a disjunctive model (Roy and Sussman 1964). This kind of modeling allows for solving the problem by deciding about the relative order among operations that share the same resources, instead of considering for every operation all its possible starting times. We propose here to use the following model for the JSSO which is an extension to that used in (Sierra, Mencía, and Varela 2013) for the JSO where all operators can assist any operation and the precedence relations define a number of jobs.

A problem instance is represented by a directed graph $G = (V, E \cup D \cup I \cup O)$ where:

- Each node in V represents either a task in $\mathcal{T}$, or one of the fictitious tasks with null processing time; namely, starting tasks for each operator $o \in \mathcal{O}$, and the the dummy operations *start* and *end*.
- E is the set of arcs of the task graph also called *conjunctive arcs*. $P(v)$ and $S(v)$ will denote the sets of tasks which are predecessors and successors of v respectively in the task graph.
- D is the set of *disjunctive arcs* which represent capacity constraints of the machines. D is partitioned into subsets D_j with $D = \cup_{j=1,\ldots,q} D_j$. D_j includes an arc (v, w) for each pair of operations requiring the machine j.
- O is the set of *operator arcs* and includes two types of arcs: one arc (u, v) for each pair of operations of the problem such that $\mathcal{O}_u \cap \mathcal{O}_v \neq \emptyset$, and arcs (o, u) for each operator node o and task u such that $o \in \mathcal{O}_u$.
- The set I includes arcs connecting node *start* to each node $o \in O$ and arcs connecting each task without successors in the task graph to the node *end*.

Arcs are weighted with the processing time of the task at the outgoing node.

From this representation, building a schedule may be viewed as a process of fixing disjunctive and operator arcs. A disjunctive arc between operations u and v gets fixed when either (u, v) or (v, u) is selected and so the other one discarded. If the operator arc (o, u) is fixed, the task u is assisted by o, and consequently all arcs (o', u) for o' other than o are discarded. Also, if the operator arc (u, v) is fixed then u and v are assisted by the same operator, u before v. In this case, the operator arc (v, u) and the remaining operator arcs connecting u or v to operations assisted by operators other than o are discarded. So, discarding both arcs (u, v) and (v, u) means that u and v will be assisted by different operators.

A feasible schedule S is represented by an acyclic subgraph of G, of the form $G_S = (V, E \cup F \cup I \cup Q)$, where $F \subset D$ expresses the processing order of tasks on the machines and $Q \subset O$ expresses the sequences of tasks that are assisted by each operator. In other words:

- $F = \cup_{j=1,\ldots,q} F_j$, $F_j \subset D_j$ such that $(u, v) \in F_j$ iff $m_u = m_v$ and u is processed before v in S. So, F_j represents the machine sequence or processing order of tasks in the machine j.
- $Q = \cup_{o=1,\ldots,p} Q_o$, where Q_o represents the operator sequence of o, and includes the arcs (o, u), (o, v) and (u, v) for each pair of operations assisted by the operator o such that u is processed before v. Each operation u must be included in one and only one operator sequence.

A *critical path* is a longest cost path in G_S from node *start* to node *end*. The *head* r_v of an operation v is the cost of the longest path from node *start* to node v and defines a lower bound for st_v. For most regular objective functions, in particular for the makespan, taking $st_v = r_v$ produces the optimal value restricted to the processing ordering defined by the solution graph. In this case, the cost of a critical path is the makespan of the schedule S denoted $C_{max}(S)$.

Figure 1 shows a solution graph for a problem instance with 7 tasks, 3 machines and 2 operators.

A partial schedule is given by a subgraph of G where some of the disjunctive and operator arcs have not fixed yet. In such a schedule $PM(v)$ denotes the disjunctive predecessors of v; $w \in PM(v)$ means that $m_w = m_v$ and that the disjunctive arc (w, v) was fixed (analogously, $SM(v)$ denotes the disjunctive successors of v). $PO(v)$ denotes the operator predecessors of v, i.e., $w \in PO(v)$ if the operator arc (w, v) was fixed (analogously, $SO(v)$ are the operator successors of v).

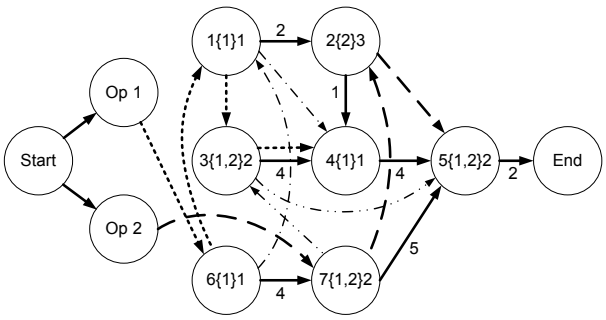


Figure 1: A solution graph for a problem instance with 7 tasks, 3 machines and 2 operators. Each task node includes the task number, the set of skilled operators and the machine using the notation $u\mathcal{O}_u m_u$. Dotted and dashed arcs represent operator sequences starting in an operator node, while dotted-dashed arcs represent machine sequences. For the sake of clarity, only the arcs between consecutive nodes in the relations are represented and only the cost of the arcs in the task graph are displayed. Every arc should be labeled with the processing time of the task at the outgoing node. The makespan is 19 and corresponds to the cost of the critical path ($start$ Op1 6 7 3 4 5 end).

Schedule Generation Schemes

In this section we propose a new schedule generation scheme for the JSSO which is a generalization of the well-known $G\&T$ algorithm proposed in (Giffler and Thompson 1960) for the classic JSP and also of the $OG\&T$ algorithm proposed in (Sierra, Mencía, and Varela 2013) for the JSO. The new schedule builder is denoted $SOG\&T$ (Skilled Operators $G\&T$).

The $SOG\&T$ Algorithm

This algorithm iterates over n steps and in each iteration one of the tasks is scheduled following an order compatible with the partial ordering defined by the task graph. So, by the time the task u is scheduled, all tasks $v \in P(u)$ have already been scheduled. At this time, u is allocated an operator $o_u = o \in \mathcal{O}_u$ and a starting time st_u such that it is scheduled after all the scheduled tasks that require the same machine m_u or that were allocated the same operator o_u, i.e., it is an $appending$ scheme.

Let SC be the set of scheduled operations before the current iteration and let $G_{SC} = (V, E \cup H \cup I \cup R)$ be the partial solution graph built so far. For all $u, v \in SC$ such that $m_u = m_v$, either (u, v) or (v, u) are in H. Also, for all $u \in SC$, (o, u) is in R for some $o \in \mathcal{O}$, and if (o, v) is also in R then either (u, v) or (v, u) are in R as well.

The *set of eligible operations* in the current iteration is defined as:

$$A = \{v; v \notin SC, P(v) \subseteq SC\} \tag{1}$$

A includes the first unscheduled tasks in the task graph. In principle, each task in A is a candidate to be scheduled next and it can be assisted by any of the operators skilled to do it. So, the *set of scheduling options* is defined as:

$$\mathcal{A} = \{(u, o); u \in A, o \in \mathcal{O}_u\} \tag{2}$$

As the set $\mathcal{A}$ may be very large, we will only consider options in a subset $\mathcal{X} \subseteq \mathcal{A}$. If the option $(u, o) \in \mathcal{X}$ is selected for the next scheduling step, then the lowest starting time of u is given by

$$st_u(o) = \max\{st_x + p_x, st_y + p_y, st_z + p_z\} \tag{3}$$

where

- x is the task in $P(u)$ with the largest completion time; i.e.,

$$x = argmax\{st_v + p_v; v \in P(u)\}, \tag{4}$$

- y is the last scheduled task in the machine required by u and
- z is the last scheduled task assisted by the operator o.

The option (u, o) establishes the time interval $[st_u(o), st_u(o) + p_u)$ for processing u under the assistance of the operator o.

Algorithm 1 shows the general structure of the schedule builder $SOG\&T$. It starts from a set A containing the tasks with no predecessors in the task graph and iterates along n steps. In each iteration an option (u, o) is taken non deterministically from a subset $\mathcal{X}$ of the scheduling options. The task u is scheduled at the time $st_u(o)$ given by exp. (3) and assisted by the operator o. Then the partial schedule built so far, G_{SC}, and the set A are updated accordingly for the next iteration. At last, after n iterations, G_{SC} represents a complete schedule.

Algorithm 1: Schedule builder $SOG\&T$. It builds a feasible schedule in n steps.

Data: A JSSO problem instance $\mathcal{P}$
Result: A feasible schedule for $\mathcal{P}$
$A = \{u \in \mathcal{T}; \neg\exists(w, u) \in E\}$;
for *i=1 to n* **do**
  $\mathcal{A} = \{(u, o); u \in A, o \in \mathcal{O}_u\}$;
  $\mathcal{X}$ = a subset of $\mathcal{A}$;
  choose $(u, o) \in \mathcal{X}$ non deterministically;
  set $st_u = st_u(o)$ and $o_u = o$;
  add u to SC and update G_{SC};
  $A = \{v; v \notin SC, P(v) \subseteq SC\}$;
end
return *the built schedule* G_{SC};

It is clear that any schedule built by Algorithm 1 is feasible. Also, depending on the subset of options considered, the algorithm will generate schedules in different search spaces. We will only consider here dominant search spaces; i.e., spaces containing at least one optimal schedule under the objective function considered, in our case the makespan.

Search spaces

The simplest way to get a dominant search space is taking $\mathcal{X} = \mathcal{A}$ in Algorithm 1. In spite of being dominant, as it is proved below, the search space generated by Algorithm 1 considering all options in $\mathcal{A}$ at each iteration may be very large, even for small instances, as we have mentioned. Fortunately, it is possible to further reduce the search space without loss of dominance.

Let us now consider the option (v^*, o^*) with the earliest completion time among the options in $\mathcal{A}$; i.e.

$$(v^*, o^*) = \arg\min\{st_u(o) + p_u; (u, o) \in \mathcal{A}\} \quad (5)$$

Let $C^* = st_{v^*}(o^*) + p_{v^*}$. We define the sets of options $\mathcal{A}'$ and $\mathcal{B}$ as follows.

$$\mathcal{A}' = \{(u, o) \in \mathcal{A}; st_u(o) < C^*\} \quad (6)$$

$$\mathcal{B} = \{(u, o) \in \mathcal{A}'; (m_u = m_{v^*} \vee o = o^*)\} \quad (7)$$

The set $\mathcal{A}'$ reduces the scheduling options to the options in $\mathcal{A}$ having a starting time lower than C^*. So, each option in $\mathcal{A}'$ establishes a starting time for a task in the interval $[T_{\mathcal{A}'}, C^*)$ where

$$T_{\mathcal{A}'} = \arg\min\{st_u(o); (u, o) \in \mathcal{A}\}. \quad (8)$$

Then, $\mathcal{B}$ further restricts the number of options by filtering those involving a machine other than m_{v^*} and an operator other than o^* at the same time. So, the set $\mathcal{B}$ contains the options for tasks in A that would require either the machine m_{v^*} or the operator o^* at some time in the interval $[T_{\mathcal{B}}, C^*)$ if they were chosen in the current iteration, where

$$T_{\mathcal{B}} = \arg\min\{st_u(o); (u, o) \in \mathcal{B}\}, \quad (9)$$

being $T_{\mathcal{B}} \geq T_{\mathcal{A}'}$ as the option that establishes the value of $T_{\mathcal{A}'}$ in expression (8) may not be included in $\mathcal{B}$.

The following result establishes that taking $\mathcal{X} = \mathcal{B}$ makes the search space dominant.

Proposition 1. *In at least one of the best schedules that can be eventually reached from the current iteration, one of the operations in A is scheduled in accordance with and option in $\mathcal{B}$.*

Proof. Let S be one schedule that can be eventually built from the current iteration such that none of the tasks in A is scheduled in accordance with an option in $\mathcal{B}$. In S, m_{v^*} and o^* are idle over the interval $[st_{v^*}(o^*), C^*)$. Then v^* could be rescheduled in accordance with the option (v^*, o^*); i.e., starting at $st_{v^*}(o^*)$ and assisted by o^*. As none of the remaining operations has to be delayed and $st_{v^*}(o^*) \leq st_{v^*}$, then $C_{max}(S') \leq C_{max}(S)$. So, if S is one of the best schedules, then S' is also one of the best schedules. □

Corollary 1. *There is a sequence of non deterministic choices of options, each one from the set $\mathcal{B}$ calculated in each iteration, that allows Algorithm 1 to reach an optimal schedule.*

Proof. It follows trivially from Proposition 1 considering the initial iteration in which none of the tasks is scheduled. □

So, due to the fact that $\mathcal{B} \subseteq \mathcal{A}' \subseteq \mathcal{A}$, taking $\mathcal{X} = \mathcal{A}'$ or $\mathcal{X} = \mathcal{A}$ makes the search space dominant as well.

Remark 1. *Regarding the sets $\mathcal{A}'$ and $\mathcal{B}$ is important to make the following observation. As we have mentioned, if in a given iteration the chosen option (u, o) is selected from $\mathcal{A}'$, then $st_u \in [T_{\mathcal{A}'}, C^*)$, while if it is selected from $\mathcal{B}$, then $st_u \in [T_{\mathcal{B}}, C^*)$, with $T_{\mathcal{A}'} \leq T_{\mathcal{B}}$. This fact may have consequences on the idle times of the machines and operators and so on the average makespan of the schedules. So, as a consequence of the potentially larger mean idle times of the schedules generated from $\mathcal{B}$, due to the difference $T_{\mathcal{B}} - T_{\mathcal{A}'}$, the average of these schedules may be larger than the average makespan of the schedules generated from $\mathcal{A}'$.*

Also, if the option (u, o) is selected from $\mathcal{A}$, then $st_u \in [T_{\mathcal{A}'}, C)$, where C is expected to be larger than C^*, so the average makespan of the schedules generated from $\mathcal{A}$ is expected to be larger than that of the schedules generated from $\mathcal{A}'$ or $\mathcal{B}$ as $C - C^*$ is likely greater than $T_{\mathcal{B}} - T_{\mathcal{A}'}$.

Summary of Search Spaces and Schedule Builders

In this section we have introduced the $SOG\&T$ as a generic schedule builder and we have seen how it may be adapted to search in three different dominant search spaces characterized by the sets of options $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$. Abusing the language we will denote these spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$ respectively.

Genetic Algorithm for the JSSO

Genetic algorithms have been successfully applied to scheduling problems such as JSP (Mattfeld 1995; Bierwirth 1995) or JSO (Mencía et al. 2014). In these cases, the problems were defined by a set of jobs, what allowed the GAs to use the efficient encoding based on permutations repetition (Bierwirth 1995). In the JSSO, as it was defined here, this encoding cannot be used due to the arbitrary precedence relations. So, in principle, we opted to use a more conventional coding schema as single permutations of tasks.

At the same time, in JSP and JSO no information about operators needs to be included in the chromosome; in the first case no operators exist, while in the second all of them are skilled to assist any operation and so they may be in fact considered as a cumulative resource of capacity p. However, in the JSSO, it is reasonable to express operator preferences for the tasks in the chromosomes in order to get an appropriate coding schema.

In the following subsections, we detail the main components of the proposed GA for the JSSO; namely, the coding schema, the decoding algorithm, the genetic operators and the general structure of the GA used.

Coding Schema

We propose here a coding schema for the JSSO where a chromosome consists of two permutations of symbols. The first one is the *task sequence* which is a conventional permutation of the numbers $1 \ldots n$, while the second is the *operator sequence* and is given by a permutation with repetition of the symbols $1 \ldots p$ of size n and represents operator preferences. Notice that some operators could appear several times in the chromosome and also some operator may be absent.

For example the following two permutations represent a feasible chromosome for the instance considered in Figure 1.

$$\begin{matrix}(6,7,5,4,1,2,3)\\(1,2,2,2,1,1,1)\end{matrix} \quad (10)$$

This encoding should be understood in such a way that if task u appears before task v in the first permutation, then u should be preferably scheduled before v. At the same time, the second permutation represents priorities for the operators to be allocated to tasks. In order to avoid that all tasks see the same order of operators, the first operator for a task will be that in the same position as the task in the first permutation and then the remaining ones are taken following the chromosome as a circular structure. An important observation is that the task ordering and the operators' allocation in the schedule does not only depend on the chromosome, but also on the decoding algorithm, as we will see in the next section.

One of the most interesting properties of this encoding is that any solution can be encoded into a chromosome representing the same machine orderings and operator allocations. At the same time, it is simple and so it allows for designing efficient genetic operators for crossover and mutation. In principle, the only inconvenience comes from the fact that the number of replicas of a symbol in the operator sequence is variable and so an operator could disappear from the chromosome. However, this problem can be easily solved by means of some heuristic repairing as we will see.

Decoding Algorithm

Decoding algorithms map chromosomes to feasible solutions. To this aim, we use the $SOG\&T$ algorithm presented above, exploiting the information encoded in the chromosomes to guide the search. This algorithm is issued and in each iteration the option $(u, o) \in \mathcal{X}$ that is the "leftmost" in the chromosome, i.e., that fulfills the following two conditions, is chosen

- u is the leftmost in the task sequence of the chromosome among all tasks appearing in some option in $\mathcal{X}$, and
- o is the first operator skilled to assist u in the preference list defined by the operator sequence, among the operators appearing in some option in $\mathcal{X}$ for the operation u. If the operator of none of the options for u in $\mathcal{X}$ is present in the operator sequence, then the operator o with $(u, o) \in \mathcal{X}$ that allows u to start earliest is selected.

In the experimental study we will consider all the possibilities above for the set of options $\mathcal{X}$; namely, $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$. In the three cases, the genetic algorithm searches over dominant search spaces and so it has the chance to reach an optimal schedule.

We illustrate how the decoding algorithm works by means of an example. Let us consider the chromosome in (10). It contains the tentative orderings (6,4,1) and (7,5,3) for machines 1 and 2 respectively, which are inconsistent due to the fact that the task 4 must be processed after the task 1, and the task 5 must be processed after the tasks 7 and 3, in accordance with the precedences expressed by the task graph. In spite of that, the chromosome is feasible as the decoding algorithm will build a feasible schedule from it in which some of the tentative orderings and operator preferences will hold, while others will not. If we consider the set $\mathcal{X} = \mathcal{A}$, in the first iteration the scheduling options would be $\{(1,1),(3,1),(3,2),(6,1)\}$; in all four cases the starting time would be 0. So, according to the chromosome, the task scheduled in this step would be 6, as it is the leftmost one in the chromosome among 1, 3, and 6, and the operator assigned would be 1 (as this is the only option). In the next iteration, the options would be $\{(3,2)\}$ with starting time 0 and $\{(1,1),(3,1),(7,1),(7,2)\}$ with starting time 4. In accordance with the chromosome, the chosen option would be (7,2). In the third iteration the options would be $\{(3,1),(3,2)\}$ with starting time 9 and $\{(1,1)\}$ with starting time 4, and the chosen option would be (1,1). This way, the tentative ordering (4,1) in the chromosome is not kept in the schedule. Finally, we will have the schedule of Fig. 1. It is important to remark that with $\mathcal{X} = \mathcal{A}'$ or $\mathcal{X} = \mathcal{B}$, we could reach different schedules from the same chromosome.

Genetic Operators

We build on conventional two point crossover and single mutation operators, and start with an initial population of chromosomes generated at random following a uniform distribution. Also, in order to better translate characteristics from parents to offsprings, we propose to code back the structure of the schedule into the chromosome. This operation is expected to produce small changes in the chromosome and so we refer to it as weak Lamarckian evolution.

We will use a double-chromosome variant of the classic order crossover (OX) which translates the subsequence of symbols between two cutting points from one parent to the offspring and the relative order of the remaining values from the second. OX may be a good option for the JSSO due to the fact that relevant characteristics, as processing order of tasks on the machines or priorities of operators for assisting the tasks, may be transmitted from parents to offsprings. To avoid a strong disruptive effect of the crossover, the cutting points will be the same in both sequences (tasks and operators).

Also, we will consider single mutation (SM) by swapping two consecutive positions at random. This seems to be appropriate as it may produce small changes. As before, the same positions will be swapped in both sequences. Moreover, to get different distributions of operators in the operator sequences, we will also use an operator mutation (OM) which will change the value in a location of the operator sequence at random. When a chromosome is mutated, one of SM or OM is chosen with probability 0.5.

Genetic Algorithm Structure

We use here a rather conventional genetic algorithm with generational replacement. In order to avoid premature convergence, we opted not to use the classic roulette wheel selection combined with unconditional replacement. Instead, in the selection phase all chromosomes are organized into pairs at random, then each pair undergoes crossover and mutation. After this, the offsprings are evaluated and the schedules coded back into the chromosomes as described in the

Algorithm 2: Genetic Algorithm.

Data: A JSSO problem instance $\mathcal{P}$ and a set of parameters $(P_c, P_m, \#gen, \#popsize, \mathcal{X})$
Result: A feasible schedule for $\mathcal{P}$
Generate and evaluate the initial population $P(0)$;
for *t=1 to #gen-1* **do**
  Selection: organize the chromosomes in $P(t-1)$ into pairs at random ;
  Recombination: mate each pair of chromosomes and mutate the two offsprings in accordance with P_c and P_m;
  Evaluation: evaluate the resulting chromosomes considering the search space $\mathcal{X}$ and code back the schedules into the chromosomes;
  Replacement: make a tournament selection among every two parents and their offsprings to generate $P(t)$;
end
return *the best schedule built so far*;

section above. Finally, the new population is obtained by means of tournament selection keeping the best two individuals among every two parents and their two offsprings. The algorithm requires 5 parameters: crossover and mutation probabilities (P_c and P_m), number of generations ($\#gen$), population size ($\#popsize$) and the search space considered $\mathcal{X}$. Algorithm 2 shows the main steps of the genetic algorithm.

Experimental Study

To assess the performance of the proposed GA and compare it with the state of the art, we have conducted an experimental study. We considered two sets of instances[1]: firstly a set of instances derived from the well-known FT10 instance for the classic JSP with $n = 100$ tasks and $q = 10$ machines and, on the other hand, a set of instances used in (Agnetis, Murgia, and Sbrilli 2014).

The instances in the first set were generated considering different values for p (5, 7 and 9 operators) and different probabilities, Pr, that an operator can assist one task (0.2 and 0.6). Five instances were generated from each pair (Pr, p) and five more from each p taking Pr as 0.2 or 0.6 at random for each task. So we have 45 instances in all. These instances are denoted 2_5_1, 2_5_2, ..., 2/6_5_1, etc.

The second set includes 50 instances organized in 5 groups with 10 instances each. Each group is defined by the values of n, p and q ranging in the sets $\{100, 150, 200\}$, $\{10, 15, 20\}$ and $\{15, 30, 50\}$ respectively. The processing times are uniformly distributed in $[1, 100]$. The topology of the task graph was inspired in real-life assembly trees and was distributed in a number of branches, between 3 and 6. The sets of operators skilled for each operation were generated at random and the workload of the machines was roughly balanced. We obtained detailed results from MSB-DOS and Cplex from a personal communication from the authors of (Agnetis, Murgia, and Sbrilli 2014). These results are summarized in Table 2.

[1]The instances and more details of the experimental study are available at http://www.di.uniovi.es/iscop (Repository).

Table 1: Summary of results from GA on the sets of instances derived from the FT10, considering the search spaces $\mathcal{A}'$ and $\mathcal{B}$. The results are averaged for each subset of 5 instances. Times are given in seconds.

Sets			$\mathcal{A}'$			$\mathcal{B}$		
#	Pr	p	Best	Avg.	Time	Best	Avg.	Time
1	2	5	1281.8	1309.3	8.7	1361.4	1397.9	7.2
2	2	7	1109.8	1142.4	8.6	1183.6	1223.5	6.9
3	2	9	1022.0	1051.5	9.5	1123.8	1151.7	7.5
4	6	5	1164.0	1187.6	14.0	1235.6	1265.0	10.9
5	6	7	1009.0	1030.1	16.7	1081.6	1110.7	12.6
6	6	9	972.2	995.6	19.4	1024.2	1046.8	14.5
7	2/6	5	1203.4	1235.7	11.5	1306.4	1349.2	9.1
8	2/6	7	1053.2	1078.0	13.5	1167.4	1196.8	10.0
9	2/6	9	985.6	1016.6	13.7	1111.4	1132.6	10.3
Average			1089.0	1116.3	12.9	1177.3	1208.2	9.9

Evaluation of the Genetic Algorithm

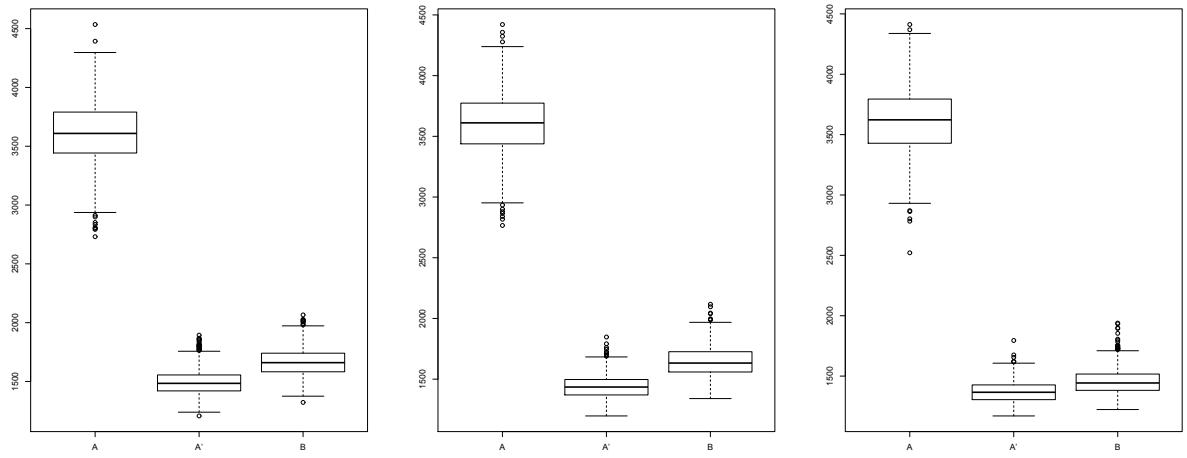
We evaluated different options in the proposed GA, in particular we considered different search spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$, and coding the schedule back into the chromosome or not. The evaluation was done on the instances in the first set. We have chosen a rather conventional parameter setting: $P_c = 1$[2], $P_m = 0.1$, $\#popsize = 100$, $\#gen = 1000$. The target machine was Intel Core i7-3770K 3.50 GHz. 12 GB RAM and the algorithm was coded in C++.

Firstly, we evaluated the average quality of the schedules in the three search spaces. To do this, we generated 1000 random solutions in each subset. Figure 2 shows the distribution of makespan outcomes for each set of schedules. As we can observe, schedules sampled from the space $\mathcal{A}$ are clearly much worse than the schedules from the other two spaces; and schedules from $\mathcal{A}'$ are better than those from $\mathcal{B}$. So, from these results, the space $\mathcal{A}'$ seems to be good for a successful evolution and convergence of the GA.

In order to visualize the evolution of the GA, we show in Figure 3 the convergence pattern of the GA for three instances considering the search spaces $\mathcal{A}'$ and $\mathcal{B}$ with and without the coding-back option. In view of these results, the coding-back option is good in all cases and decoding in $\mathcal{A}'$ is better than decoding in $\mathcal{B}$. We conducted the same experiments considering the set $\mathcal{A}$ and the results were much worse as it was expected from the first experiments.

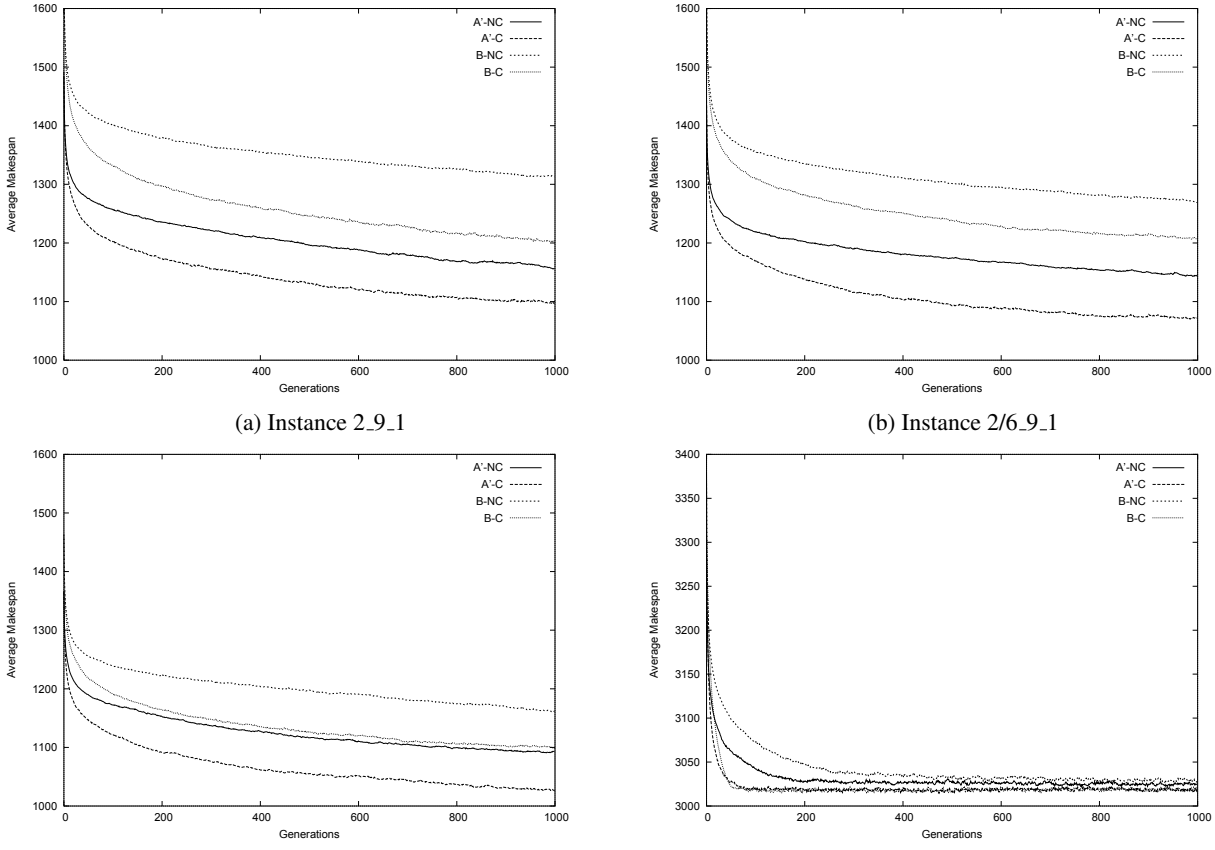
To further assess the differences between decoding in the subsets $\mathcal{A}'$ and $\mathcal{B}$, we solved the 45 instances with the coding-back option. In this case, each instance was solved 10 times and the best and average of the solutions reached

[2]Taking $P_c < 1$ makes that some pairs of chromosomes are not mated and in this case the offsprings are the same as their parents if they are not mutated, so the best parent is chosen twice in the replacement phase, what may contribute to premature convergence.



(a) Instance 2_9_1 (b) Instance 2/6_9_1 (c) Instance 6_9_1

Figure 2: Summary of results from 1000 random schedules for the instances 2_9_1, 2/6_9_1 and 6_9_1 derived from the FT10 in the spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$.



(a) Instance 2_9_1 (b) Instance 2/6_9_1

(c) Instance 6_9_1 (d) Instance $100/10/15\_1$

Figure 3: Convergence of the GA for the instances 2_9_1, 2/6_9_1 and 6_9_1 derived from the FT10 and the instance $100/10/15\_1$ from (Agnetis, Murgia, and Sbrilli 2014), combining the search spaces $\mathcal{A}'$ and $\mathcal{B}$ and the coding (C) and non-coding (NC) back options. Each plot represents the evolution of the mean makespan of the population averaged for 10 runs.

Table 2: Summary of results from MSB-DOS and Cplex averaged for the 10 instances in each of the 10 sets. $\#Opt.$ is the number of instances in each subset which are optimally solved. Time is given in seconds. For GA, the values reported are the average values of the 10 runs for each instance.

Sets				MSB-DOS			CPLEX		GA		
#	Tasks(n)	Operators(p)	Machines(q)	GAP%	Time	#Opt.	Time	#Op.	GAP%	Time	#Opt.
1	100	10	15	0.78	106.44	5.00	106.21	10.00	0.06	1.84	8.00
2	150	15	30	0.20	8.28	6.00	190.54	10.00	0.00	2.87	10.00
3	150	15	50	0.03	9.58	9.00	129.28	10.00	0.00	2.84	10.00
4	200	15	30	0.42	25.62	6.00	755.85	10.00	0.00	3.64	9.70
5	200	20	30	0.41	23.43	6.00	913.32	10.00	0.00	4.24	10.00
Average				0.37	34.67	6.40	419.04	10.00	0.01	3.09	9.54

in all runs were recorded. These results, averaged for each subset of 5 instances, together with the average time taken to solve one instance in each run, are summarized in Table 1. These results confirm that $\mathcal{A}'$ is the best choice as the makespan with $\mathcal{A}'$ is about 8.2% better than it is with $\mathcal{B}$. This is quite reasonable from Remark 1 above. As we can observe, in all cases the values from $\mathcal{A}'$ are better than those from $\mathcal{B}$; even the average values from $\mathcal{A}'$ are better than the best values from $\mathcal{B}$. Also, the time taken with $\mathcal{B}$ is about 23.25% lower than it is with $\mathcal{A}'$, what is natural as $\mathcal{A}'$ offers more options in each step of the schedule builder and so looking for the leftmost one in the chromosome takes more time. We have also registered the results from $\mathcal{A}'$ by the time taken with $\mathcal{B}$, and the difference is still significant, it only drops from 8.2% to 7.7% in favor of $\mathcal{A}'$.

Comparison with Other Methods

In (Agnetis, Murgia, and Sbrilli 2014) the authors proposed two heuristic methods, MSB-DOS and STA-OMSB and they also experimented with Cplex 12.2 on a MILP formulation they proposed. The authors report results from this study averaged for each subset of instances showing clearly that MSB-DOS is the best of the two heuristics proposed. Their target machine was AMD Athlon II 2.70 GHz using Matlab 2009b. Cplex was given a time limit of 3600 s. and was able to find optimal solutions for all the 50 instances.

Table 2 summarizes the results produced by MSB-DOS, Cplex and our GA. GA was parameterized as indicated above, with the only difference that the number of generations given was 250 instead of 1000. The reason for this is that GA needs less generations to converge due to the topology of the precedence graph. This fact can be observed in Figure 3(d), which represents the convergence patterns for the instance $100/10/15\_1$; after generation 250 the GA hardly converges any more with the four combinations. In this example, there are not significant differences between spaces $\mathcal{A}'$ and $\mathcal{B}$ when the coding-back option is considered.

As we can observe, Cplex can reach optimal solutions for all the instances with an average time of 419.04s. MSB-DOS takes much lower time in average, 34.67s, and obtains optimal solutions for 6.4 instances in average for each subset with an average gap of 0.37%. Regarding GA, it is able to obtain optimal solutions in the 10 runs for all instances in three sets, 2,3 and 5; and in average for 8 and 9.7 instances in the remaining two sets. Overall, it was able to obtain the optimal solution in 477 of the 500 runs and in at least one of the 10 runs for 49 of the 50 instances. The only instance not solved optimally at least once was instance 8 of the set 1 where the best makespan reached was 4537, being the optimum 4534 (GA was able to find an optimal solution in 1000 generations). The average gap in percent was 0.01 and the average time taken 3.09s indicating that, despite the differences on the target machines, GA is faster than the methods proposed in the literature, and finds much better solutions than any other approximate approach.

Conclusions and Future Work

We have seen that the well-known $G\&T$ schedule builder proposed in (Giffler and Thompson 1960) for the classic JSP problem may be extended to solve JSSO problem proposed in (Agnetis, Murgia, and Sbrilli 2014), in a similar way as it was previously extended to other problems such as the $EG\&T$ for the JSP with Sequence Dependent Setup times (Artigues, Lopez, and Ayache 2005), the $OG\&T$ for the JSO (Sierra, Mencía, and Varela 2013) or the $fG\&TSGS$ for the JSP with fuzzy processing times (González Rodríguez, Vela, and Puente 2007; Palacios et al. 2014).

Much in the same way as $G\&T$, $EG\&T$, $fG\&TSGS$ and $OG\&T$ were exploited in (Mattfeld 1995), (Vela, Varela, and González 2010), (González Rodríguez, Vela, and Puente 2007) and (Mencía et al. 2014) respectively, $SOG\&T$ has been exploited here to devise a decoder which is the core of the GA proposed to solve the JSSO. The success of this algorithm relies not only in the capability of the $SOG\&T$ to generate schedules in different search spaces, but also in the proposed coding schema that encodes candidate solutions by means of a conventional permutation of tasks and a permutation with repetition of operators.

As future work, we plan to design local search algorithms which will be combined with the GA. To this end, we will devise neighborhood structures for the JSSO from the proposed disjunctive model. In principle, we will try to extend the structures proposed in (Dell' Amico and Trubian 1993; Laarhoven, Aarts, and Lenstra 1992) and consider new structures aiming at exploring changes in the assignment of operators. Furthermore, we will tackle the JSSO in the framework of heuristic search using $SOG\&T$ as branching schema, in this case $\mathcal{B}$ will be likely the best option.

In this setting, the disjunctive graph model could help to devise consistent heuristics and dominance rules. We expect it to work well when using the proper search strategy as, for example, in (Mencía, Sierra, and Varela 2013) for the JSO. It will also be interesting to consider and evaluate constraint programming approaches on the JSSO, both general frameworks as CPLEX CP Optimizer (Laborie 2009), or more specific algorithms such as the successful Solution Guided Search (Beck, Feng, and Watson 2011).

Acknowledgments

This research has been supported by the Spanish Government under research projects TIN2010-20976-C02-02 and TIN2013-46511-C2-2-P, and by the Principality of Asturias under project FICYT2013 - COF13-035. Carlos Mencía is supported by SFI grant BEACON (09/IN.1/I2618). We are grateful to the anonymous reviewers for their insightful comments.

References

Adams, J.; Balas, E.; and Zawack, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34(3):391–401.

Agnetis, A.; Flamini, M.; Nicosia, G.; and Pacifici, A. 2011. A job-shop problem with one additional resource type. *J. Scheduling* 14(3):225–237.

Agnetis, A.; Murgia, G.; and Sbrilli, S. 2014. A job shop scheduling problem with human operators in handicraft production. *International Journal of Production Research* 52(13):3820–3831.

Artigues, C.; Lopez, P.; and Ayache, P. 2005. Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. *Annals of Operations Research* 138:21–52.

Beck, J. C.; Feng, T. K.; and Watson, J. 2011. Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing* 23(1):1–14.

Bierwirth, C. 1995. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spectrum* 17:87–92.

Dell' Amico, M., and Trubian, M. 1993. Applying tabu search to the job-shop scheduling problem. *Annals of Operational Research* 41:231–252.

Giffler, B., and Thompson, G. L. 1960. Algorithms for solving production scheduling problems. *Operations Research* 8:487–503.

González Rodríguez, I.; Vela, C. R.; and Puente, J. 2007. A memetic approach to fuzzy job shop based on expectation model. In *Proceedings of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2007*, 692–697. London: IEEE.

Laarhoven, P. J. M. v.; Aarts, E. H. L.; and Lenstra, J. K. 1992. Job shop scheduling by simulated annealing. *Operations Research* 40(1):pp. 113–125.

Laborie, P. 2009. IBM CP Optimizer for detailed scheduling illustrated on three problems. In van Hoeve, W.-J., and Hooker, J., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 148–162.

Mattfeld, D. C. 1995. *Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling*. Springer-Verlag.

Mencía, R.; Sierra, M. R.; Mencía, C.; and Varela, R. 2014. A genetic algorithm for job-shop scheduling with operators enhanced by weak lamarckian evolution and search space narrowing. *Natural Computing* 13(2):179–192.

Mencía, C.; Sierra, M. R.; and Varela, R. 2013. An efficient hybrid search algorithm for job shop scheduling with operators. *International Journal of Production Research* 51(17):5221–5237.

Palacios, J. J.; Vela, C. R.; Rodríguez, I. G.; and Puente, J. 2014. Schedule generation schemes for job shop problems with fuzziness. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic*, 687–692.

Roy, B., and Sussman, B. 1964. Les problemes d'ordonnancements avec contraintes disjonctives. Notes DS no. 9 bis, SEMA, Paris.

Sierra, M.; Mencía, C.; and Varela, R. 2013. New schedule generation schemes for the job-shop problem with operators. *Journal of Intelligent Manufacturing* 1–15.

Vela, C. R.; Varela, R.; and González, M. A. 2010. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics* 16(2):139–165.